REMARKS

The enclosed is responsive to the Examiner's Final Office Action mailed on August 10, 2007 and is being filed pursuant to a Request for Continued Examination (RCE) as provided for under 37 CFR 1.114. At the time the Examiner mailed the Office Action claims 1-33 were pending. By way of the present response the Applicants have: 1) amended claims 1, 12 and 23; 2) added no new claims. As such, claims 1-33 are now pending. The Applicants respectfully request reconsideration of the present application and the allowance of all claims now represented.

The Examiner rejected independent claims 1, 12 and 23 under 35 U.S.C. 103(a) as being unpatentable over U.S. Pub. Pat. No. 2002/0073063 (hereinafter "Faraj"), in view of U.S. Pat. No. 6,961,918 B2 (hereinafter "Garner"). The Applicant respectfully disagrees with the Examiner for a number of reasons.

According to the Examiner's reasoning, Faraj discloses all elements of independent claim 1 except for program execution across application servers, databases and/or external systems. See, Examiner's Office Action mailed 8/10/07, pp. 2-3. The Applicant strongly disagrees with this reasoning. Most troubling to the Applicant is that the Examiner believes Faraj discloses "modifying bytecode associated with the one or more application components". See, Examiner's Office Action mailed 8/10/07, p. 2.

The Applicant respectfully submits that this is simply not true. Faraj does not teach a complete working system, at least in conjuction with the other aspects of Faraj that the Examiner has cited as teaching the Applicant's claim elements, that employs bytecode modification. Faraj, to the contrary, explicitly disclaims that his invention uses bytecode modification. The following excerpts from Faraj clearly demonstrate this fact.

> [0010] A second approach to automating instrumentation of Java applications is the post-processing of compiled Java byte code. This approach uses byte code manipulation libraries that enable the insertion and modification of byte code in a Java .class file so that the Java code generates an execution trace at runtime. This approach has the following two disadvantages. Firstly, modifying byte code potentially creates Java security problems. When a JAR file (Java archive file) is signed using JDK software tools, each file in the archive is given a digest entry in the archive's manifest. The digest values are hashes or encoded representations of the contents of the files as they were at the time of signing, and they will change if the file itself changes. Verifying a signed JAR file means that the digests of each of its files is to be re-computed and then compared with the digests recorded in the manifest to ensure that the contents of the JAR file haven't changed since it was signed. Hence, byte code modification will result in difficulties where the application to be traced is in a signed JAR file. The application will not pass the security verification step as a result of the modification to the byte files made for tracing purposes.
>
> [0011] In addition, modifying the byte code means that the resulting code can no longer be easily debugged. Debuggers expect a certain match between Java source files and their corresponding compiled class files. When the byte code in a .class file is modified, there is no longer a match between the source code line numbers and the new .class files.
>
> [0012] Thus, as may be seen from the above description, there are significant limitations inherent in the prior art approaches to generating a runtime trace for Java applications.
>
> [0015] It is therefore desirable to have a mechanism for generating VM-independent traces of Java applications **without byte-code manipulation**, the mechanism having a problem determination tool able to be used with different applications.

Thus, to the extent that the Examiner has used certain features of the invention taught by Firaj to cover a number of the Applicant's claim elements, it is clear these features were not meant to be included in a

system that uses bytecode modification. Therefore Firaj is at least deficient with respect to its ability to teach bytecode modification in combination with the other elements of the Firaj system relied upon by the Examiner.

The applicant hereby brings the Examiner's attention two other references that the Applicant believes actually disclose bytecode modification and are more pertinent to the claims of the present application. These are:

1) U.S Pub'd App. No. 2004/0123279 ("Boykin")

2) U.S. Pub'd App. No. 2005/0039171 ("Avakian")

The Applicant discusses these two references immediately below.

Boykin

Boykin discloses a bytecode modification system, however, Boykin is different from the claimed invention in at least two respects. Firstly, Boykin modifies the bytecode during loading of their respective classfiles while the Applicant's claimed invention recites the act of modifying bytecode before respective classfiles are loaded. The Examiner is invited to compare, the following excerpts of Boykin (where "hooks" are understood to be bytecode modifications)

> At class load time, an injector determines whether a loaded class has any instrumentation locations as predetermined by information in the registry. If so, the injector inserts hooks in the loaded class.
> Boykin, abstract (emphasis added).

> Using the probe location information, injector component 214 inserts hooks into original class files 202 at class load time to create manageable classes 216 comprising hooks . . .
> Boykin, para. [0037] (emphasis added).

> During the class load process, the class loader provides an indication, e.g., class load event notification 408, to injector 410, which then injects hooks into the classes.
> Boykin, para. [0045] (emphasis added).

with the following comments that appear in the Applicant's specification

> . . . the bytecode modifier module 352 may modify the bytecode 350 prior to runtime . . .
> Applicant's Specification, para. [0053] (emphasis added).

> According to the depiction of Figure 17, the method associated with each of information structures 1703_1 through 1703_N "registers" with the dispatch unit 1702 by sending the dispatch unit 1702 its class name, the names of each of its methods and the types of arguments of its methods. By way of example, as illustrated in Figure 17, modified classfile 1701_1 sends the dispatch unit 1702 its class name and the names and argument types of each of its method 1705 . . . In an embodiment registration occurs as a consequence of a modified classfile being loaded.
> Applicant's Specification, paras. [0144], [0145], [0146] (emphasis added).

Comparison of these excerpts clearly reveals that the Boykin reference discloses actual modification of byte code at runtime during classloading while the Applicant's specification discloses actual modification of byte code before runtime and therefore prior to classloading.

Secondly, as the excerpt from the Applicant's specification immediately above reveals, during loading of classfile at runtime, the Applicant's specification discloses that an already modified classfile "registers" its name and its method names to a dispatcher. The Boykin reference is also different in this respect in that Boykin only discloses the passing of a classfile's identity (and not its methods' names) to an injector during class loading. See, e.g., Boykin para. [0046] ("Injector 410 can

determine whether to inject a hook into a recently loaded class by querying the probe registry 416, e.g., by using an identifier of the recently loaded class, which may be provided to injector 410 through class load event notification 408).

Avakian

The Avakian reference does not disclose a dispatching process as claimed by the applicant. See, e.g., Applicant's Specification, Fig. 5a and corresponding discussion. Rather than using a dispatch process in which instrumented methods first invoke a dispatcher before interfacing directly to a plug-in, Avakian discloses that instrumented methods directly call to a plug-in interface (specifically, ExecCallback interface 36) without referral to a dispatcher beforehand. See, e.g., Avakian, para. [0059] ("The inserted instrumentation code associated with one or methods of the [modified classfile] C' can invoke the ExecCallback interface 36 . . . "); id, para. [0061] ("The execCallback interface 36 enables various types of monitoring tools (hereinafter "plug-in instruments") to be plugged into the interface and receive information when classes are executed."); id., para. [0095] (describing the functional calls that can be made to the ExecCallback interface); id., Fig. 7 (showing an instrumented classfile's method making direct calls to the ExecCallback interface through methods described in para. [0095] (specifically "methodEntry" 702, "reportArg" 708, "methodException" 716, "methodExit" 706).

CONCLUSION

Because the Applicant has demonstrated the patentability of all pending independent claims, the Applicant respectfully submits that all pending claims are allowable. The Applicant's silence with respect to the dependent claims should not be construed as an admission by the Applicant that the Applicant is complicit with the Examiner's rejection of these claims. Because the Applicant has demonstrated the patentability of the independent claims, the Applicant need not substantively address the theories of rejection applied to the dependent claims. Moreover, where the Applicant has failed to address a specific independent claim element alleged by the Examiner to be covered by prior art, such failure should not be viewed as an admission by the Applicant that the Applicant accepts or agrees with the Examiner's reasoning.

In the further interests of efficiency, the Applicant reserves the right under MPEP 2144.03.C to cause the Examiner to find in the prior art subject matter to which the Examiner has taken Official Notice at a later time in the prosecution of the present case when the subject matter of such prior art is actually at issue.

Applicant respectfully submits that all redetections have been overcome and that all pending claims are in condition for allowance.

If there are any additional charges, please charge them to our Deposit Account Number 02-2666. If a telephone conference would facilitate the prosecution of this application, the Examiner is invited to contact Thomas C. Webster at (408) 720-8300.

Respectfully submitted,
BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP

Date: 11/13

Robert B. O'Rourke
Reg. No.: 46,972

12400 Wilshire Boulevard
Seventh Floor
Los Angeles, CA 90025-1026
(408) 720-8300